PATENT
112056-0180
P01-1786

UNITED STATES PATENT APPLICATION

of

Emily Eng

Andy Kahn

and

John Edwards

for a

TECHNIQUE FOR INCREASING THE NUMBER OF PERSISTENT CONSISTENCY POINT IMAGES IN A FILE SYSTEM

TECHNIQUE FOR INCREASING THE NUMBER OF PERSISTENT CONSISTENCY POINT IMAGES IN A FILE SYSTEM

FIELD OF THE INVENTION

The present invention relates to file systems and, more specifically, to a technique for increasing the number of persistent consistency point images of a file system.

BACKGROUND OF THE INVENTION

A storage system typically comprises one or more storage devices into which information may be entered, and from which information may be obtained, as desired. The storage system includes a storage operating system that functionally organizes the system by, *inter alia*, invoking storage operations in support of a storage service implemented by the system. The storage system may be implemented in accordance with a variety of storage architectures including, but not limited to, a network-attached storage environment, a storage area network and a disk assembly directly attached to a client or host computer. The storage devices are typically disk drives organized as a disk array, wherein the term "disk" commonly describes a self-contained rotating magnetic media storage device. The term disk in this context is synonymous with hard disk drive (HDD) or direct access storage device (DASD).

Storage of information on the disk array is preferably implemented as one or more storage "volumes" of physical disks, defining an overall logical arrangement of disk space. The disks within a volume are typically organized as one or more groups, wherein each group may be operated as a Redundant Array of Independent (or Inexpensive) Disks (RAID). Most RAID implementations enhance the reliability/integrity of data storage through the redundant writing of data "stripes" across a given number of physical disks in the RAID group, and the appropriate storing of redundant information (parity) with re-

spect to the striped data. The physical disks of each RAID group may include disks configured to store striped data (*i.e.*, data disks) and disks configured to store parity for the data (*i.e.*, parity disks). The parity may thereafter be retrieved to enable recovery of data lost when a disk fails. The term "RAID" and its various implementations are well-known and disclosed in *A Case for Redundant Arrays of Inexpensive Disks (RAID)*, by D. A. Patterson, G. A. Gibson and R. H. Katz, Proceedings of the International Conference on Management of Data (SIGMOD), June 1988.

The storage operating system of the storage system may implement a high-level module, such as a file system, to logically organize the information stored on the disks as a hierarchical structure of directories, files and blocks. For example, each "on-disk" file may be implemented as set of data structures, *i.e.*, disk blocks, configured to store information, such as the actual data for the file. These data blocks are organized within a volume block number (vbn) space that is maintained by the file system. The file system organizes the data blocks within the vbn space as a "logical volume"; each logical volume may be, although is not necessarily, associated with its own file system. The file system typically consists of a contiguous range of vbns from zero to *n*, for a file system of size *n*-1 blocks.

A known type of file system is a write-anywhere file system that does not overwrite data on disks. If a data block is retrieved (read) from disk into a memory of the storage system and "dirtied" (i.e., updated or modified) with new data, the data block is thereafter stored (written) to a new location on disk to optimize write performance. A write-anywhere file system may initially assume an optimal layout such that the data is substantially contiguously arranged on disks. The optimal disk layout results in efficient access operations, particularly for sequential read operations, directed to the disks. An example of a write-anywhere file system that is configured to operate on a storage system is the Write Anywhere File Layout (WAFL™) file system available from Network Appliance, Inc., Sunnyvale, California.

The storage operating system may further implement a storage module, such as a RAID system, that manages the storage and retrieval of the information to and from the

disks in accordance with input/output (I/O) operations. The RAID system is also responsible for parity operations in the storage system. Note that the file system only "sees" the data disks within its vbn space; the parity disks are "hidden" from the file system and, thus, are only visible to the RAID system. The RAID system typically organizes the RAID groups into one large "physical" disk (i.e., a physical volume), such that the disk blocks are concatenated across all disks of all RAID groups. The logical volume maintained by the file system is then "disposed over" the physical volume maintained by the RAID system.

The storage system may be configured to operate according to a client/server model of information delivery to thereby allow many clients to access the directories, files and blocks stored on the system. In this model, the client may comprise an application, such as a database application, executing on a computer that "connects" to the storage system over a computer network, such as a point-to-point link, shared local area network, wide area network or virtual private network implemented over a public network, such as the Internet. Each client may request the services of the file system by issuing file system protocol messages (in the form of packets) to the storage system over the network. By supporting a plurality of file system protocols, such as the conventional Common Internet File System (CIFS) and the Network File System (NFS) protocols, the utility of the storage system is enhanced.

When accessing a block of a file in response to servicing a client request, the file system specifies a vbn that is translated at the file system/RAID system boundary into a disk block number (dbn) location on a particular disk (disk, dbn) within a RAID group of the physical volume. Each block in the vbn space and in the dbn space is typically fixed, e.g., 4k bytes (KB), in size; accordingly, there is typically a one-to-one mapping between the information stored on the disks in the dbn space and the information organized by the file system in the vbn space. The (disk, dbn) location specified by the RAID system is further translated by a disk driver system of the storage operating system into a sector (or similar granularity) on the specified disk.

The requested block is then retrieved from disk and stored in a buffer cache of the memory as part of a buffer tree of the file. The buffer tree is an internal representation of blocks for a file stored in the buffer cache and maintained by the file system. Broadly stated, the buffer tree has an inode at the root (top-level) of the file. An inode is a data structure used to store information, such as metadata, about a file, whereas the data blocks are structures used to store the actual data for the file. The information contained in an inode may include, e.g., ownership of the file, access permission for the file, size of the file, file type and references to locations on disk of the data blocks for the file. The references to the locations of the file data are provided by pointers, which may further reference indirect blocks that, in turn, reference the data blocks, depending upon the quantity of data in the file. Each pointer may be embodied as a vbn to facilitate efficiency among the file system and the RAID system when accessing the data on disks.

The file system, such as the write-anywhere file system, maintains information about the geometry of the underlying physical disks (e.g., the number of blocks in each disk) in the storage system. The RAID system provides the disk geometry information to the file system for use when creating and maintaining the vbn-to-disk,dbn mappings used to perform write allocation operations. The file system maintains block allocation data structures, such as an active map, a space map, a summary map and snapmaps. These mapping data structures describe which blocks are currently in use and which are available for use and are used by a write allocator of the file system as existing infrastructure for the logical volume.

Specifically, the snapmap denotes a bitmap file describing which blocks are used by a snapshot. The write-anywhere file system (such as the WAFL file system) has the capability to generate a *snapshot* of its active file system. An "active file system" is a file system to which data can be both written and read or, more generally, an active store that responds to both read and write I/O operations. It should be noted that "snapshot" is a trademark of Network Appliance, Inc. and is used for purposes of this patent to designate a persistent consistency point (CP) image. A persistent consistency point image (PCPI) is a space conservative, point-in-time read-only image of data accessible by name that provides a consistent image of that data (such as a storage system) at some previous time.

More particularly, a PCPI is a point-in-time representation of a storage element, such as an active file system, file or database, stored on a storage device (e.g., on disk) or other persistent memory and having a name or other identifier that distinguishes it from other PCPIs taken at other points in time. A PCPI can also include other information (metadata) about the active file system at the particular point in time for which the image is taken. The terms "PCPI" and "snapshot" may be used interchangeably through out this patent without derogation of Network Appliance's trademark rights.

The write-anywhere file system supports (maintains) multiple snapshots that are generally created on a regular schedule. Each snapshot refers to a copy of the file system that diverges from the active file system over time as the active file system is modified. Each snapshot is a restorable version of the storage element (e.g., the active file system) created at a predetermined point in time and, as noted, is "read-only" accessible and "space-conservative". Space conservative denotes that common parts of the storage element in multiple snapshots share the same file system blocks. Only the differences among these various snapshots require extra storage blocks. The multiple snapshots of a storage element are not independent copies, each consuming disk space; therefore, creation of a snapshot on the file system is instantaneous, since no entity data needs to be copied. Read-only accessibility denotes that a snapshot cannot be modified because it is closely coupled to a single writable image in the active file system. The closely coupled association between a file in the active file system and the same file in a snapshot obviates the use of multiple "same" files. In the example of a WAFL file system, snapshots are described in *TR3002 File System Design for a NFS File Server Appliance* by David Hitz et al., published by Network Appliance, Inc. and in U.S. Patent No. 5,819,292 entitled *Method for Maintaining Consistent States of a File System and For Creating User-Accessible Read-Only Copies of a File System*, by David Hitz *et al.*, each of which is hereby incorporated by reference as though full set forth herein.

The active map denotes a bitmap file describing which blocks are used by the active file system. As described prior, a snapshot may contain metadata describing the file system as it existed at the point in time that the image was taken. In particular, a snapshot captures the active map as it existed at the time of snapshot creation; this file is also

known as the snapmap for the snapshot. Note then that a snapmap denotes a bitmap file describing which blocks are used by a snapshot. The summary map denotes a file that is an inclusive logical OR bitmap of all snapmaps. By examining the active and summary maps, the file system can determine whether a block is in use by either the active file system or any snapshot. The space map denotes a file including an array of numbers that describe the number of storage blocks used in a block allocation area. In other words, the space map is essentially a logical OR bitmap between the active and summary maps to provide a condensed version of available "free block" areas within the vbn space. Examples of snapshot and block allocation data structures, such as the active map, space map and summary map, are described in U.S. Patent Application Publication No. US2002/0083037 A1, titled *Instant Snapshot*, by Blake Lewis *et al.* and published on June 27, 2002, which application is hereby incorporated by reference.

Fig. 1 is a schematic block diagram of an exemplary on-disk storage structure 100 of a logical volume of a storage system. As noted, a logical volume is typically associated with a file system and comprises data blocks organized within a vbn space. Each logical volume (hereinafter "volume") has a file system information (fsinfo) block that is preferably stored at a fixed location within, e.g., a RAID group. Fsinfo block 105 is the root of the on-disk storage structure 100, illustratively at vbns 1 and 2. When loading the volume, the storage operating system accesses those vbns to acquire the fsinfo block 105.

The fsinfo block 105 includes a variety of metadata that describes the state of the file system; also included in the fsinfo block 105 is an inode for an inode file 110. All inodes of the write-anywhere file system are organized into the inode file 111. Like any other file, the inode of the inode file is the root of the buffer tree that describes the location of blocks o the file. As such, the inode of the inode file may directly reference (point to) data blocks 107 of the inode file 111 or may reference indirect blocks 106 of the inode file 111 that, in turn, reference data blocks of the inode file. In this example, the inode for the inode file 110 includes an exemplary buffer tree comprising a plurality of inode file indirect blocks 106 that, in turn, point to inode file data blocks 107. Within each data block of the inode file are inodes 112, each of which serves as the root of a file. Among the inodes of the inode file 110, there are inodes for special metadata files, such as an ac-

tive map 115, a summary map 120, a space map 125, a root directory 140 and a metadata directory 145. All user files in the file system are organized under the root directory 140, while various metadata files associated with the file system are stored under the metadata directory 145.

The inode file may further include inodes that reference a plurality of snapshots 130, 135. These snapshot inodes are the root level inodes of snapshots (PCPIs) of the active file system. Each volume has special reserved inode numbers within its vbn space; a plurality of those inode numbers (e.g., 31) is reserved for PCPIs. When a PCPI is generated of the active file system, a copy of the inode for the inode file is generated (hereinafter the "snapshot root") and assigned one of the reserved PCPI inode numbers. Thus, to access a PCPI at a particular point in time, the storage operating system accesses the appropriate snapshot root of the PCPI.

A noted disadvantage of such an on-disk storage structure is a limitation on the number PCPIs (e.g., 31) that may be maintained with the file system. As a result, a system administrator (user) may be forced to modify PCPI creation and/or retention schedules to avoid exhausting the available number of maintainable PCPIs. This limitation may prove burdensome and, possibly, costly depending upon the need for additional PCPI capacity. The present invention is directed to alleviating this limitation.

SUMMARY OF THE INVENTION

The present invention overcomes the disadvantages of the prior art by providing an on-disk storage arrangement that increases the number of persistent consistency point images (PCPIs) that may be maintained for a volume of a storage system. The on-disk storage arrangement comprises a novel volume information (volinfo) block representing the root of the volume; the volinfo block is stored at predefined locations on disk and comprises various system wide configuration data. According to the invention, the volinfo block further comprises a data structure configured to provide a level of indirection that increases the number of PCPIs maintainable by a file system executing on the storage system. To that end, the data structure may be organized as an array of pointers,

wherein each pointer references a data block comprising a snapshot root, thereby enabling efficient access to each PCPI maintained by the file system.

In the illustrative embodiment, the volume comprises data blocks organized within a volume block number (vbn) space maintained by the file system. The array is embodied as a vbn lookup table having a plurality of entries, wherein each entry comprises a vbn pointer configured to point to (reference) a file system information (fsinfo) block within the volume. The fsinfo block contains information that specifies a layout of the file system. Each entry of the vbn lookup table is indexed by an identifier assigned to each PCPI; notably, entry zero holds a vbn pointer to the "active" file system. Thus, one of the fsinfo blocks referenced by the vbn lookup table is associated with the active file system, while the remaining fsinfo blocks are associated with PCPIs of the active file system.

Advantageously, the novel vbn lookup table enables efficient access to information describing the active file system and, illustratively, 255 PCPIs. This feature of the invention permits an illustrative eight-fold increase in the number of PCPIs maintainable by the file system. Additional PCPIs may be maintained in the storage system by configuring the vbn lookup table to provide further levels of indirection. For example, the entries of the vbn lookup table may be configured to reference indirect fsinfo blocks that, in turn, reference "direct" fsinfo blocks. Therefore by expanding the number of levels of indirection, any number of PCPIs may be maintained with the file system.

BRIEF DESCRIPTION OF THE DRAWINGS

The above and further advantages of the invention may be better understood by referring to the following description in conjunction with the accompanying drawings in which like reference numerals indicate identical or functionally similar elements:

Fig. 1, already described, is a schematic block diagram of an on-disk storage structure of a volume of a storage system;

Fig. 2 is a schematic block diagram of an environment including a storage system that may be added advantageously used with a present invention;

Fig. 3 is a schematic block diagram of a storage operating system that be advantageously used with a present invention;

Fig. 4 is a schematic block diagram of an inode that may be advantageously used with the present;

Fig. 5 is a schematic block diagram of an on-disk storage arrangement of a volume in accordance with an illustrative embodiment of the present invention;

Fig. 6 is a schematic block diagram of an exemplary volume information block in accordance with the embodiment with the present information; and

Fig. 7 is a schematic block diagram of an on-disk storage arrangement of a volume in accordance with an alternate embodiment of the present invention.

DETAILED DESCRIPTION OF AN ILLUSTRATIVE EMBODIMENT

A. Storage System Environment

Fig. 2 is a schematic block diagram of an environment 200 including a storage system 220 that may be advantageously used with the present invention. The storage system is a computer that provides storage service relating to the organization of information on storage devices, such as disks 230 of a disk array 260. The storage system 220 comprises a processor 222, a memory 224, a network adapter 226 and a storage adapter 228 interconnected by a system bus 225. The storage system 220 also includes a storage operating system 300 that preferably implements a high-level module, such as a file system, to logically organize the information as a hierarchical structure of directories, files and special types of files called virtual disks (hereinafter "blocks") on the disks.

In the illustrative embodiment, the memory 224 comprises storage locations that are addressable by the processor and adapters for storing software program code. A portion of the memory may be further organized as a "buffer cache" 270 for storing certain data structures associated with the present invention. The processor and adapters may, in turn, comprise processing elements and/or logic circuitry configured to execute the software code and manipulate the data structures. Storage operating system 300, portions of which are typically resident in memory and executed by the processing elements, func-

tionally organizes the system 220 by, *inter alia*, invoking storage operations executed by the storage system. It will be apparent to those skilled in the art that other processing and memory means, including various computer readable media, may be used for storing and executing program instructions pertaining to the inventive technique described herein.

The network adapter 226 comprises the mechanical, electrical and signaling circuitry needed to connect the storage system 220 to a client 210 over a computer network 240, which may comprise a point-to-point connection or a shared medium, such as a local area network. Illustratively, the computer network 240 may be embodied as an Ethernet network or a Fibre Channel (FC) network. The client 210 may communicate with the storage system over network 240 by exchanging discrete frames or packets of data according to pre-defined protocols, such as the Transmission Control Protocol/Internet Protocol (TCP/IP).

The client 210 may be a general-purpose computer configured to execute applications 212. Moreover, the client 210 may interact with the storage system 220 in accordance with a client/server model of information delivery. That is, the client may request the services of the storage system, and the system may return the results of the services requested by the client, by exchanging packets 250 over the network 240. The clients may issue packets including file-based access protocols, such as the Common Internet File System (CIFS) protocol or Network File System (NFS) protocol, over TCP/IP when accessing information in the form of files and directories. Alternatively, the client may issue packets including block-based access protocols, such as the Small Computer Systems Interface (SCSI) protocol encapsulated over TCP (iSCSI) and SCSI encapsulated over Fibre Channel (FCP), when accessing information in the form of blocks.

The storage adapter 228 cooperates with the storage operating system 300 executing on the system 220 to access information requested by a user (or client). The information may be stored on any type of attached array of writable storage device media such as video tape, optical, DVD, magnetic tape, bubble memory, electronic random access memory, micro-electro mechanical and any other similar media adapted to store information, including data and parity information. However, as illustratively described

herein, the information is preferably stored on the disks 230, such as HDD and/or DASD, of array 260. The storage adapter includes input/output (I/O) interface circuitry that couples to the disks over an I/O interconnect arrangement, such as a conventional high-performance, FC serial link topology.

Storage of information on array 260 is preferably implemented as one or more storage "volumes" that comprise a collection of physical storage disks 230 cooperating to define an overall logical arrangement of volume block number (vbn) space on the volume(s). Each logical volume is generally, although not necessarily, associated with its own file system. The disks within a logical volume/file system are typically organized as one or more groups, wherein each group may be operated as a Redundant Array of Independent (or Inexpensive) Disks (RAID). Most RAID implementations, such as a RAID-4 level implementation, enhance the reliability/integrity of data storage through the redundant writing of data "stripes" across a given number of physical disks in the RAID group, and the appropriate storing of parity information with respect to the striped data. An illustrative example of a RAID implementation is a RAID-4 level implementation, although it should be understood that other types and levels of RAID implementations may be used in accordance with the inventive principles described herein.

B. Storage Operating System

To facilitate access to the disks 230, the storage operating system 300 implements a write-anywhere file system that cooperates with virtualization modules to "virtualize" the storage space provided by disks 230. The file system logically organizes the information as a hierarchical structure of named directories and files on the disks. Each "on-disk" file may be implemented as set of disk blocks configured to store information, such as data, whereas the directory may be implemented as a specially formatted file in which names and links to other files and directories are stored. The virtualization modules allow the file system to further logically organize information as a hierarchical structure of blocks on the disks that are exported as named logical unit numbers (luns).

In the illustrative embodiment, the storage operating system is preferably the NetApp® Data ONTAP™ operating system available from Network Appliance, Inc.,

Sunnyvale, California that implements a Write Anywhere File Layout (WAFL™) file system. However, it is expressly contemplated that any appropriate storage operating system may be enhanced for use in accordance with the inventive principles described herein. As such, where the term "WAFL" is employed, it should be taken broadly to refer to any file system that is otherwise adaptable to the teachings of this invention.

Fig. 3 is a schematic block diagram of the storage operating system 300 that may be advantageously used with the present invention. The storage operating system comprises a series of software layers organized to form an integrated network protocol stack or, more generally, a multi-protocol engine that provides data paths for clients to access information stored on the storage system using block and file access protocols. The protocol stack includes a media access layer 310 of network drivers (e.g., gigabit Ethernet drivers) that interfaces to network protocol layers, such as the IP layer 312 and its supporting transport mechanisms, the TCP layer 314 and the User Datagram Protocol (UDP) layer 316. A file system protocol layer provides multi-protocol file access and, to that end, includes support for the Direct Access File System (DAFS) protocol 318, the NFS protocol 320, the CIFS protocol 322 and the Hypertext Transfer Protocol (HTTP) protocol 324. A VI layer 326 implements the VI architecture to provide direct access transport (DAT) capabilities, such as RDMA, as required by the DAFS protocol 318.

An iSCSI driver layer 328 provides block protocol access over the TCP/IP network protocol layers, while a FC driver layer 330 receives and transmits block access requests and responses to and from the storage system. The FC and iSCSI drivers provide FC-specific and iSCSI-specific access control to the blocks and, thus, manage exports of luns to either iSCSI or FCP or, alternatively, to both iSCSI and FCP when accessing the blocks on the storage system. In addition, the storage operating system includes a storage module embodied as a RAID system 340 that manages the storage and retrieval of information to and from the volumes/disks in accordance with I/O operations, and a disk driver system 350 that implements a disk access protocol such as, *e.g.*, the SCSI protocol.

Bridging the disk software layers with the integrated network protocol stack layers is a virtualization system that is implemented by a file system 380 interacting with

virtualization modules illustratively embodied as, *e.g.*, vdisk module 390 and SCSI target module 370. The vdisk module 390 is layered on the file system 380 to enable access by administrative interfaces. The SCSI target module 370 is disposed between the FC and iSCSI drivers 328, 330 and the file system 380 to provide a translation layer of the virtualization system between the block (lun) space and the file system space, where luns are represented as blocks.

The file system is illustratively a message-based system that provides logical volume management capabilities for use in access to the information stored on the storage devices, such as disks. That is, in addition to providing file system semantics, the file system 380 provides functions normally associated with a volume manager. These functions include (i) aggregation of the disks, (ii) aggregation of storage bandwidth of the disks, and (iii) reliability guarantees, such as mirroring and/or parity (RAID). The file system 380 illustratively implements the WAFL file system (hereinafter generally the "write-anywhere file system") having an on-disk format representation that is block-based using, e.g., 4 kilobyte (KB) blocks and using index nodes ("inodes") to identify files and file attributes (such as creation time, access permissions, size and block location). The file system uses files to store metadata describing the layout of its file system; these metadata files include, among others, an inode file. A file handle, i.e., an identifier that includes an inode number, is used to retrieve an inode from disk.

Broadly stated, all inodes of the write-anywhere file system are organized into the inode file. Each logical volume (hereinafter "volume") has a file system information (fsinfo) block specifies the layout of data in the file system and includes the inode of the "inode file," the file contains all other inodes of the file system. The inode of the inode file may directly reference (point to) blocks of the inode file or may reference indirect blocks of the inode file that, in turn, reference direct blocks of the inode file. Within each direct block of the inode file are inodes, each of which serves as the root of a buffer tree of a file.

Operationally, a request from the client 210 is forwarded as a packet 250 over the computer network 240 and onto the storage system 220 where it is received at the net-

work adapter 226. A network driver (of layer 310 or layer 330) processes the packet and, if appropriate, passes it on to a network protocol and file access layer for additional processing prior to forwarding to the write-anywhere file system 380. Here, the file system generates operations to load (retrieve) the requested data from disk 230 if it is not resident "in core", *i.e.*, in the buffer cache 270. If the information is not in the cache, the file system 380 indexes into the inode file using the inode number to access an appropriate entry and retrieve a logical vbn. The file system then passes a message structure including the logical vbn to the RAID system 340; the logical vbn is mapped to a disk identifier and disk block number (disk,dbn) and sent to an appropriate driver (e.g., SCSI) of the disk driver system 350. The disk driver accesses the dbn from the specified disk 230 and loads the requested data block(s) in buffer cache 270 for processing by the storage system. Upon completion of the request, the storage system (and operating system) returns a reply to the client 210 over the network 240.

It should be noted that the software "path" through the storage operating system layers described above needed to perform data storage access for the client request received at the storage system may alternatively be implemented in hardware. That is, in an alternate embodiment of the invention, a storage access request data path may be implemented as logic circuitry embodied within a field programmable gate array (FPGA) or an application specific integrated circuit (ASIC). This type of hardware implementation increases the performance of the storage service provided by storage system 220 in response to a request issued by client 210. Moreover, in another alternate embodiment of the invention, the processing elements of adapters 226, 228 may be configured to offload some or all of the packet processing and storage access operations, respectively, from processor 222, to thereby increase the performance of the storage service provided by the system. It is expressly contemplated that the various processes, architectures and procedures described herein can be implemented in hardware, firmware or software.

As used herein, the term "storage operating system" generally refers to the computer-executable code operable to perform a storage function in a storage system, e.g., that manages data access and may, in the case of a file server, implement file system semantics. In this sense, the ONTAP software is an example of such a storage operating

system implemented as a microkernel and including the WAFL layer to implement the WAFL file system semantics and manage data access. The storage operating system can also be implemented as an application program operating over a general-purpose operating system, such as UNIX® or Windows NT®, or as a general-purpose operating system with configurable functionality, which is configured for storage applications as described herein.

In addition, it will be understood to those skilled in the art that the inventive technique described herein may apply to any type of special-purpose (e.g., file server, filer or multi-protocol storage appliance) or general-purpose computer, including a standalone computer or portion thereof, embodied as or including a storage system 220. An example of a multi-protocol storage appliance that may be advantageously used with the present invention is described in U.S. Patent Application Serial No. 10/215,917 titled, *Multi-Protocol Storage Appliance that Provides Integrated Support for File and Block Access Protocols*, by Vijayan Rajan, *et al.* Moreover, the teachings of this invention can be adapted to a variety of storage system architectures including, but not limited to, a network-attached storage environment, a storage area network and disk assembly directly-attached to a client or host computer. The term "storage system" should therefore be taken broadly to include such arrangements in addition to any subsystems configured to perform a storage function and associated with other equipment or systems.

In the illustrative embodiment, a file is represented in the write-anywhere file system by an inode data structure adapted for storage on the disks 230. Fig. 4 is a schematic block diagram of an inode 400, which preferably includes a metadata section 410 and a data section 450. The information stored in the metadata section 410 of each inode 400 describes the file and, as such, includes the type (e.g., regular, directory, virtual disk) 412 of file, the size 414 of the file, time stamps (e.g., access and/or modification) 416 for the file and ownership, i.e., user identifier (UID 418) and group ID (GID 420), of the file.

The data section 450 of an on-disk inode may contain file data or pointers, the latter referencing 4KB data blocks on disk used to store the file data. Each pointer is preferably a logical vbn to facilitate efficiency among the file system and the RAID sys-

tem 340 when accessing the data on disks. Given the restricted size (e.g., 128 bytes) of the inode, file data having a size that is less than or equal to 64 bytes is represented, in its entirety, within the data section of that inode. However, if the file data is greater than 64 bytes but less than or equal to 64KB, then the data section of the inode (e.g., a first level inode) comprises up to 16 pointers, each of which references a 4KB block of data on the disk.

Moreover, if the size of the data is greater than 64KB but less than or equal to 64 megabytes (MB), then each pointer in the data section 450 of the inode (e.g., a second level inode) references an indirect block (e.g., a first level block) that contains 1024 pointers, each of which references a 4KB data block on disk. For file data having a size greater than 64MB, each pointer in the data section 450 of the inode (e.g., a third level inode) references a double-indirect block (e.g., a second level block) that contains 1024 pointers, each referencing an indirect (e.g., a first level) block. The indirect block, in turn, contains 1024 pointers, each of which references a 4KB data block on disk. When accessing a file, each block of the file may be loaded from disk 230 into the buffer cache 270.

The contents of the file data residing in the level zero data blocks of the file will be interpreted differently depending on the type of file (inode) defined within the type field 412. For example, the data blocks of a directory inode contains metadata that adhere to a structure defined by the file system to describe a directory, whereas the data blocks of a regular inode may simply contain the data associated with the file.

When an on-disk inode (or block) is loaded from disk 230 into buffer cache 270, its corresponding in core structure embeds the on-disk structure. For example, the dotted line surrounding the inode 400 (Fig. 4) indicates the in core representation of the on-disk inode structure. The in core structure is a block of memory that stores the on-disk structure plus additional information needed to manage data in the memory (but not on disk). The additional information may include, e.g., a "dirty" bit 460. After data in the inode (or block) is updated/modified as instructed by, e.g., a write operation, the modified data is marked "dirty" using the dirty bit 460 so that the inode (block) can be subsequently

"flushed" (stored) to disk. The in core and on-disk format structures of the WAFL file system, including the inodes and inode file, are disclosed and described in the previously incorporated U.S. Patent No. 5,819,292 titled *Method for Maintaining Consistent States of a File System and for Creating User-Accessible Read-Only Copies of a File System* by David Hitz et al., issued on October 6, 1998.

C. Increasing the Number of PCPIs in a File System

The present invention is directed to an on-disk storage arrangement that increases the number of persistent consistency point images (PCPIs) that may be maintained by file system 380 for a volume of storage system 220. The storage arrangement comprises a novel volume information (volinfo) block representing a root of the volume; the volinfo block is stored at predefined locations on disk 230 and comprises various system wide configuration data. In particular, the volinfo block contains appropriate fields so that software, including versions of the storage operating system, may recognize that the volinfo block is not a fsinfo block. As noted above, the fsinfo block is typically located at vbns 1 and 2; as described herein, the on-disk storage arrangement of the present invention replaces the fsinfo block with the novel volinfo block at this predefined location.

Fig. 5 is a schematic block diagram of an exemplary on-disk storage arrangement 500 of a file system according to an illustrative embodiment of the present invention. The on-disk storage arrangement 500 comprises a volinfo block 600 that contains pointers to various fsinfo blocks including fsinfo block 505 representing the active files system, as well as fsinfo blocks 510 and 515 representing various PCPIs associated with the active file system. The volinfo 600 is illustratively located at vbns 1 and 2 or, in alternate embodiments, at another predetermined location on disk.

Each fsinfo block 505, 510, 515 is illustratively contained within an fsinfo file, the contents of which comprise the fsinfo block. In this example, the fsinfo block 505 for the active file system includes the inodes of the inode file for the active file system 520. The inode file for the active file system 520 includes further inodes for an active map 525, a summary map 530, a space map 535, a root directory 540 and a hidden metadata directory 545. Each additional fsinfo block, for example, fsinfo blocks 510 and 515, that

is associated with a PCPI includes the inode of the inode file for the PCPI, which in turn includes appropriate inodes for active maps and the like (not shown) for the specific PCPI.

Fig. 6 is a schematic block of an exemplary volinfo block 600 in accordance with an embodiment of the present invention. The volinfo block 600 includes an fsinfo magic field 605, an fsinfo version field 610, a file system creation time field 615, a time of most recent CP 620 and a count of a number of CPs for file system field 625. The block 600 also includes a volinfo magic field 630, a volinfo version 635, a file system options field 640, a password field 645, a legacy PCPIs field 650 and a list of PCPIs stored in block one of inode file field 655. In addition, the volinfo block includes a delete mask field 660, a create mask field 665, a first checksum field 670, an vbn array field 680 and a second checksum field 675. It should be noted that in alternate embodiments additional and/or differing fields may be included in the volinfo block 600.

The fsinfo magic field 605 stores an appropriate magic value for backward compatibility with earlier fsinfo block. Similarly, the fsinfo version field 610 is needed for backwards compatibility, and is tagged with a version number that indicates that the block is a volinfo block, not an fsinfo block. The fsinfo creation time field 615 stores a value indicating the time that the file system was created. Field 620 identifies the time when a most recent consistency point (CP) was generated and field 625 stores a count of the number of CPs performed on the file system. The volinfo magic field 630 stores a magic number identifying the data structure 600 as a volinfo block, and the volinfo version field 635 stores the version of the volinfo block. Note that the fsinfo block retains a version field, which continues to be the version that is updated to reflect any on-disk format change that is made to the file system. However, should a major change to the volinfo data structure occur, the volinfo version field 635 would be updated as well. The file system options field 640 includes a set of options for the file system and the password field 645 stores a password associated with the file system. The password may be utilized to limit certain administrative operations.

The legacy PCPI field 650 identifies those PCPIs that were created with an older on-disk format which lacked certain important metadata files (e.g., the fsinfo file). The content of field 655 identifies those PCPIs whose snapshot roots are stored in block 1 of the inode file, i.e., PCPIs that were created on a file system prior to the present invention. The delete mask field 660 and create mask field 665 identify PCPIs that are currently being deleted or created. The first checksum field 670 stores a checksum calculated on the contents of fields 605-665, whereas the second checksum field 675 stores a checksum calculated for the entire volinfo block 600.

According to the invention, the volinfo block 600 comprises a data structure configured to provide a level of indirection that increases the number of PCPIs maintainable by file system 380. To that end, the data structure may be organized as an array of pointers, wherein each pointer references a block containing a snapshot root (i.e., inode for the inode file of a PCPI), thereby enabling efficient access to each PCPI maintained by the file system. The array of pointers is contained in the vbn array field 680.

In the illustrative embodiment, the array is preferably embodied as a vbn lookup table 682 having a plurality of entries 684, wherein each entry comprises a vbn pointer 686 configured to point to (reference) a fsinfo block within the volume. As noted, the fsinfo block contains information that specifies the layout of the file system. Each entry 684 of the vbn lookup table 682 is indexed by an identifier (ID) assigned to each PCPI; notably, entry zero holds a vbn pointer 686 to the "active" file system. Thus, one of the fsinfo blocks referenced by the vbn lookup table 682 is associated with the active file system, while the remaining fsinfo blocks are associated with PCPIs of the active file system.

Advantageously, the novel vbn lookup table 682 in vbn array field 680 enables efficient access to information describing the active file system and, illustratively, 255 PCPIs. This feature of the invention permits an illustrative eight-fold increase in the number of PCPIs maintainable by the file system. Additional PCPIs may be maintainable in the storage system by configuring the vbn lookup table 682 to provide further levels of indirection. For example, the entries of the vbn lookup table may be configured to refer-

ence indirect fsinfo blocks that, in turn, reference "direct" fsinfo blocks. Therefore by expanding the number of levels of indirection, any number of PCPIs may be maintained with the file system. It should be noted that in embodiments utilizing indirect blocks, all per-PCPI state fields, e.g., delete mask 660 and create mask 665, would be located in the appropriate indirect blocks and not in the volinfo block.

Fig. 7 is a schematic block diagram of such an on-disk storage arrangement 700 in accordance with an alternate embodiment of the present invention. The on-disk storage arrangement 700 has a volinfo block 600 representing a root of a volume and including pointers to indirect fsinfo block 705 and 715. Note that the pointers are illustratively vbns and are organized as an array in vbn array field 680. Each indirect fsinfo block 705, 715 includes a vbn lookup table 710, 720 (similar to vbn lookup table 682) that maps PCPI IDs to appropriate fsinfo blocks. For example, entry 0 of the vbn lookup table 710 is indexed by PCPI ID 0 and the resulting vbn references fsinfo block 725 for the active file system. Similarly, entry 1 of the vbn lookup table 710 is indexed by PCPI ID 1 and the resulting vbn references fsinfo block 730 associated with PCPI 1.

To again summarize, the present invention is directed to an on-disk storage arrangement that increases the number of persistent consistency point images (PCPIs) that may be maintained for a volume of a storage system. The novel volume information (volinfo) block represents a root of the volume and comprises various system wide configuration data. The volinfo block further comprises a data structure configured to provide a level of indirection that increases the number of PCPIs maintainable by a file system executing on the storage system. To that end, the data structure may be organized as an array of pointers, wherein each pointer references a block containing a snapshot root, thereby enabling efficient access to each PCPI maintained by the file system.

The foregoing description has been directed to specific embodiments of this invention. It will be apparent, however, that other variations and modifications may be made to the described embodiments, with the attainment of some or all of their advantages. For instance, it is expressly contemplated that the teachings of this invention can be implemented as software, including a computer-readable medium having program in-

structions executing on a computer, hardware, firmware, or a combination thereof. Accordingly this description is to be taken only by way of example and not to otherwise limit the scope of the invention. Therefore, it is the object of the appended claims to cover all such variations and modifications as come within the true spirit and scope of the invention.

What is claimed is: